



Department of Information Systems and Computing

**BSc (Hons) Business Computing / Computer Science / Information
Systems (with Option if appropriate)**

Academic Year 2014 – 2015

CO-OPERATION OF MOBILE ROBOTICS

Nikhil Kishore

1137188

A report submitted in partial fulfilment of the requirements for the degree of
Bachelor of Science

Brunel University
Department of Information Systems and Computing
Uxbridge
Middlesex
UB8 3PH
United Kingdom
T: +44 1895 203397
F: +44 (0) 1895 251686

Abstract

Co-operation of mobile robotics is an attempt to bring into light a new and rising field where robots are programmed to work together as a species. With exponential rise in the processing power of computers as explained by Moore's Law the day is not far when we will see robots roaming around us and helping us with day today task. In fact the revolution already started. In this dissertation I would like to focus on how robots could talk to each other and work together for a common task. I have done this project in a group with my group partner Sangmok, both of us worked on different algorithms in the field of co-operative mobile robotics and studied if co-operative mobile robotic could actually be our future one day. I intended to study the same by implementing ant like biology inspired Algorithm in a finch robot. This algorithm in a finch works by allowing finch to explore new routes and by keeping track of all the routes it travelled. The recorded path could help other finches to follow the shorter path found by the first finch. Data mining is one of the popular Artificial Intelligence technique used in algorithm for using data to find the best route over time. Program aims to prevent other finches from searching the already explored space over and over again for finding the shortest route. I aim to develop an application where finch searches for the shortest route in a testing environment and see if it could succeed in finding the shorter route overtime and help other finch robots by informing about it. I will also be comparing and studying about the feasibility of combining mine and Sangmok's algorithm together. This combination might also help finch find exit more easily since sangmok's algorithm uses light source for guiding finch to the exit similar to the co-ordinates of the GPS. The project attempts to explore the area of co-operative mobile robotics through this application.

Acknowledgements

First and foremost I would like to use this opportunity to thank my parents without whose support nothing would have been possible.

Thanks Mom and Dad for supporting me and believing in me.

I would like to thank my tutor Stasha Lauria for supporting me through regular tutor meetings and guiding me throughout my work and also correcting me in places where I unknowingly committed mistakes.

I would also like to thank my second tutor Robert Macredie, for the kind feedback on my work which helped me writing my dissertation.

My group mate Sangmok also helped me and worked with me for the common topic and I would like to thank him for being there, whenever I got stuck or needed some advice.

Last but not the least I would like to thank the almighty for bringing all the luck and that little push of motivation I always needed at the right time for completing this dissertation.

I certify that the work presented in the dissertation is my own unless referenced.

Signature _____

Date _____

Total Words: 10,332

Table of Contents

Abstract	i
Acknowledgements	ii
Table of Contents	iii
List of Tables.....	v
List of Figures.....	6
1 INTRODUCTION.....	7
1.1 Aims and Objectives.....	8
1.1 Project Approach.....	8
1.2 Dissertation Outline	9
2 BACKGROUND.....	10
2.1 Co-operative Mobile Robotics.....	10
2.2 Artificial Intelligence and Algorithms.....	10
2.3 Natural Computing	12
2.4 Ants	13
3 METHODOLOGY	16
3.1 Introduction	16
3.2 UML diagrams.....	16
3.3 Risks.....	19
3.4 FINCH ALGORITHM.....	20
4 IMPLIMENTATION	22
4.1 Introduction	22
4.2 Environment	22
4.3 Path Planning.....	23
4.4 Data Interpretation.....	24
5 TEAM EFFORT.....	27
5.1 Follow Light Algorithm.....	27
5.2 Limitations.....	27
5.3 Combing Algorithms	27
6 CODE AND WORKING	29
7 EVALUATION.....	38
7.1 Introduction	38
7.2 Algorithm Testing	38
7.3 RESULT ANALYSIS.....	41

7.4	EVALUATING & TESTING GROUP WORK.....	43
8	CONCLUSIONS.....	47
8.1	Future Work.....	47
	References	49
Appendix A	Personal Reflection	50
A.1	Reflection on Project.....	50
A.2	Personal Reflection.....	51
Appendix B	Appendices	52
B.1	Data set.....	52
B.2	CODE.....	54
B.3	Running Program Instruction	55

List of Tables

Table 1 Dissertation Outline	9
Table 2 Risk.....	19
Table 3 Environment images.....	23
Table 4 Directions Table.....	24
Table 5 Console views and co-ordinate system.....	26
Table 6 Light algorithm time and movements	43
Table 7 best time and movements.....	44
Table 8 time taken in 20 iteration	45
Table 9 Result Summary.....	45

List of Figures

Figure 2-1 Expert System.....	11
Figure 2-2 ACO.....	14
Figure 3-1 Use case.....	17
Figure 3-2 Package-class diagram.....	18
Figure 3-3 Finch Working Model	21
Figure 4-1 Testing space specification	22
Figure 4-2 Cardboard Bordered space	23
Figure 4-3 Black box.....	23
Figure 4-4 console View 1.....	26
Figure 4-5- Console View 2.....	26
Figure 4-6 Finch Movement in co-ordinate System.	26
Figure 5-1 Light Follow.....	28
Figure 6-1 Random movement code	29
Figure 6-2 LEFT CASE code.....	30
Figure 6-3 updating direction code.....	31
Figure 6-4 writing path code	32
Figure 6-5 Reading path code.....	32
Figure 6-6 Reading splitting path code	33
Figure 6-7 Scanning Route code.....	34
Figure 6-8 Writing shortest distance code.....	35
Figure 6-9 Server connection code	35
Figure 6-10 Client code.....	36
Figure 6-11 Second Finch main class	37
Figure 7-1 Distance and directions file.....	38
Figure 7-2 console view of searching shortest route	39
Figure 7-3 Shortest Distance at 5 th iteration.....	39
Figure 7-4 Sum of distance upto 10 iteration	40
Figure 7-5 Sum of distance a upto 15 iteration.....	40
Figure 7-6 shortest distance at 14 th iteration	40
Figure 7-7 Sum of distance a upto 20 iteration.....	41
Figure 7-8 shortest distance at 19 th iteration	41
Figure 7-9 Directions	42
Figure 7-10 Distance vs iteration.....	43

1 INTRODUCTION

We have always seen a future with robots and many people believe that artificial intelligence and better communication could make that possible. Internet reflects the power of communication, which made world so small and accessible. My degree major is in Artificial Intelligence and Networking is my second optional module, which motivated me to choose this topic for my dissertation.

This project studies the relevance of cooperation of multiple mobile robotics through an application which tries to implement ant like biology inspired algorithm in finch robot for finding the shortest route to its destination. Project aims at bringing better robots to the society which could easily communicate with each other for solving a common task by supporting co-operative mobile robotics. This is a small attempt to study and test to see the feasibility of implementing Artificial Intelligence Algorithm in robots and see if they could actually do what is usually practiced in computer simulation. For dissertation ant like biologically inspired algorithm appeared to be a challenging algorithm to be implemented in Finch. It is challenging since mostly biological algorithms are attempted on computer simulations than on actual robots. Project aims at solving the problem of finding the shortest route through this application. Application also focusses on the importance of co-operation.

Project is implemented by using a Finch robot that needs to find the shortest route in a rectangular walled space in order to find the shortest route using my application. Managing data and finding patterns in data is part of the application of A.I called data mining. Data mining technique has been used in my application for scanning and finding improved routes. Data of distance covered by the robot is used to find improved and shorter route in this project. This shortest route found by finch could be sent to other finches who can now follow the shortest route without the need of searching for the shortest route by themselves again. Algorithm is designed to suit the needs and hardware of the Finch robot.

This is also a group project where my group partner will work on another algorithm relevant to the same field and in an effort to look at the same topic from different perspective. He will work on another algorithm and we will try to see if his algorithm can be used with my algorithm in order to make finch more co-operative with other robots and more efficient in doing the desired task.

Overall through this dissertation I would like to study the feasibility and understand how multiple robots could work together and also study about various constraints which prevents the technology from its implementation in today's world.

1.1 Aims and Objectives

Project aims at the implementation of an artificial intelligence algorithm in finch for finding the shortest route to the exit, which finch could forward to the other finch robots who can now follow the received shortest route to same exit.

Finch tries to find the exit in a controlled testing environment. Project tries to understand the application of co-operative mobile robotics in real world through this software. Program needs to fulfill the following objectives in order to achieve the aim.

Objectives:

- Finch successfully finds the exit and records the route for distance calculation.
- Finch will update the shortest distance by comparing all the distances it travelled.
- Shortest distance found in the given number of iteration will become the new shortest distance.
- Shortest distance data should be sent to another finch that could now follow the shortest route found by the first finch.
- Studying the possibility of combining my algorithm with my group partner's algorithm for improved results in same application of finding the shortest route.

1.1 Project Approach

This project will start by explaining the background theory where I'll be explaining how this project attempts to solve a real world problem and the necessary background literature required for explaining the software. Next I'll explain the methodology for achieving the aim. Later in chapter 4 and 5 I'll be explaining how my software works and how it could be combined with my group partner's algorithm. Next I'll try to explain my programming code and finally I'll analyse the results and evaluate the whole software.

Results from the test will be critically analysed. We will use the data to see if Finch succeeded in finding a shorter route with increase in iterations. We will also study from the results about the type of communication required for sending the short route data to another robot. Results will also be useful for comparing the efficiency of algorithm when compared with another algorithm done by my group mate with similar goal.

1.2 Dissertation Outline

Chapter Title	Description
1. Introduction	The chapter introduces us to the topic we are going to cover. It also clarifies the Aims and Objectives we wish to fulfil through this project.
2. Background	The chapter is about the literature review done for this project and general explanation of the topics necessary to be covered for better understanding of the developed application.
3. Research Methodology	Research Methodology explains how we are going to achieve the aims and objective. It also focusses on the approach and measuring scales used for achieving the final aim.
4. Implementation	Implementation explains in detail about how my program works and how it could be tested.
5. Team Effort	Team effort compares the two algorithms of mine and my group partner for studying the scope and benefits of combining the algorithms.
6. Code and Working	Chapter explains about the programming part of the algorithm and the working of finch.
7. Evaluation	This chapter evaluates the data and interprets it in more comprehensive form for better understanding of the program. Here results of all the tests are also discussed.
8. Conclusions	The chapter concludes the whole dissertation. This chapter also explains about the future works which could be done in the field and future of the application.

Table 1 Dissertation Outline

2 BACKGROUND

2.1 Co-operative Mobile Robotics

Co-operative Mobile Robotics is a rising field with many people and organizations already working on it. There is a rise in interest in the field due to various reasons which mostly includes studying group architecture, resource conflict, origin of co-operation, learning and geometric problems. Attempt of making robots work together and studying their behavior as a group has been done from very long time. Example is Grey Walter's along with Wiener and Shannon in mid 1940s who tested tortoise like robot which were initially meant for learning purpose. Those robots, which were equipped with basic hardware, like light sensor and touch sensor exhibited complex social behavior while responding with each other. This also gained enormous publicity (Walter J Freeman (2003)). French cybernetic researcher Pierre de Latil, 1953 regarded them as revolutionary.

This was one of the early researches in the field but mobile artificial agents are being more actively studied in a field called distributed artificial intelligence (DAI) since 1970's. (Yaniv Altshuler, Vladimir Yanovski and Israel A. Wagner, 2008) Although co-operation of mobile robotics was initially not the idea in DAI, they were more involved with software agents. Co-operation of mobile robotics actually became popular in 1980's when researchers actually practiced innovative projects like SWARM, ACTRESS, CEBOT, GOFER etc. it was just the beginning as even Cellular robotics system aka CEBOT or GOFER or ACTRESS were never practiced on more than 3 robots but they worked pretty well on simulations (Fukunaga, A.S., Kahng A.B., Meng, F., 1995). Today we have many military applications and also even in researches we see many robots which perform applications like foraging, box pushing etc. cooperatively which gives us the idea of rise in the field of co-operative multiple robotics in the following years.

2.2 Artificial Intelligence and Algorithms

Artificial Intelligence and algorithms today play a very crucial role in the rising field of robotics. It gives them brains and makes them expert in their field. DENDRAL, MYCIN etc. are few of the best classic examples of artificial intelligence, which can provide an expert level advice (M. Negnevitsky, 2005). Human mental process is too complex to be represented as an algorithm. However experts can express their knowledge in form of rules. Same rule based system is used in finch for keeping track of directions, co-ordinates and manipulating routes (Table 4). All the routes travelled are recorded in a text file by finch in a form of database for finding the best route. More the finch runs more data it has to scan and more chances it has of

finding a better route. Figure 1 explains how knowledge and database of facts combined and processed through inference engine forms an expert system out of finch robot.

Algorithms are important as it helps us know what we are doing and what we want to do. Algorithm which is implemented in finch is inspired by ants. It is a biology inspired algorithm that took its inspiration from ants. Nature inspired algorithm plays a crucial role in the evolution of the robotics field. These algorithms are explained in more detail in next section.

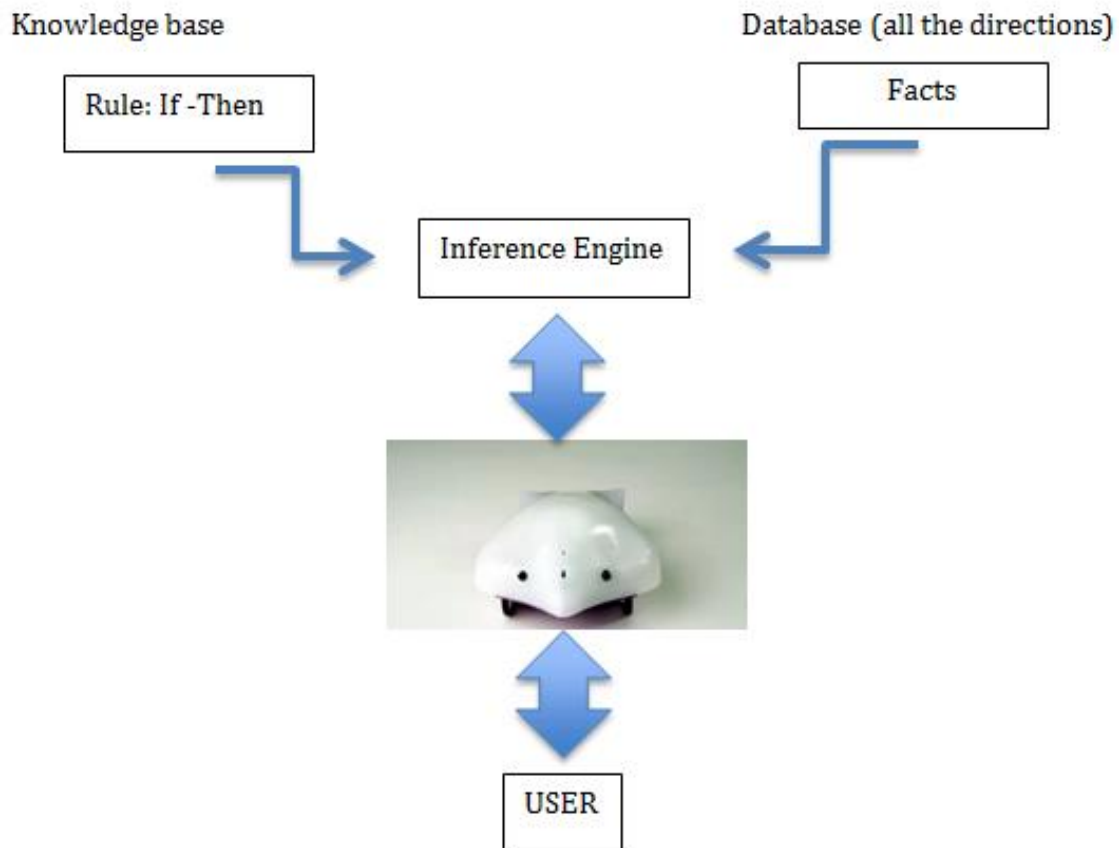


Figure 2-1 Expert System

Knowledge base is a rule based expert system which possesses the information necessary for solving the problem.

Database contains set of all the facts that is used to know which information is to be applied in the given condition. It is used to match the “if” part of the rule in knowledge base.

Inference engine connects the rules and the facts for reaching the solution.

User is user interface for integrating with the system

2.3 Natural Computing

Biology analogies are very often used in co-operative mobile robotics and they are also found pretty effective in the field. Some simple actions which we witness in our daily life could be practiced computationally, which give space to the field of natural computing. Some of the popular example of algorithms inspired from biology includes ACO where, ants when working in a group always find the shortest route to their food or John Henry Holland's genetic algorithm which is inspired by us. It takes inspiration from how chromosomes are created from genes and how different genes can provide different solutions. It also improves and finds better solutions through evolution, mutation and crossover. Children are born which are analogous to new solution and the Darwin's theory of survival of the fittest is applied in order to keep the best solutions and eliminate the rest. There are many algorithms which took their inspiration from nature. There is also a dedicated field called Artificial Life (ALife) whose ultimate goal is to understand the essential properties of life organisms by simulating them in computers. Today not only algorithms but many robots are also inspired directly from different kinds of existing species on earth. For example Boston Dynamics which is a very big and reputed company in the field of robotics have most of their robots inspired from nature like CHEETAH which actually runs at the maximum speed of 28 miles/hour and obviously inspired from cheetah or SanFlea who can make jump up to 30 feet in order to avoid obstacle by adapting the jumping strategies from grasshopper (Samuel Gibbs. 2013). But mostly existing robot works individually and co-operative mobile computing is not very often practiced.

D.J McFarland distinguished the existence of two societies in nature based on their behaviour as

- Eusocial behavior: It is found in insect species (e.g. ant colonies, bee colonies, etc.). In eusocial societies, individual agents are not very capable, but their intelligent behavior arises out of their interactions. This "cooperative" behavior is necessary for the survival of the individuals in the colonies.
- Cooperative behavior: It is found in animals (e.g. vertebrates). Unlike eusocial behavior, cooperative behavior is not motivated by innate behavior, but by an intentional desire to cooperate in order to maximize individual utility.

It is very clear that cognitive capabilities of insects individually are very limited but their complex behavior emerges from interactions among each other. Similar theory is applied by many researchers in their projects. It could also be found very useful in the case of finch robot where single finch robot may take long time for collecting path data solutions while many finches can collect more data if ran at the same time.

In my application single finch runs many times in same space to explore all possible routes for finding the best route to the exit, but if like ants I would have allowed many finches to run together the same result could have been achieved much faster saving lot of time.

Natural Computing is widely accepted and social behaviour of species in biology is one of the most important innate attribute which if successfully could be implemented in robotics could help us achieve a lot in the field.

2.4 Ants

We as kids and even as a grownups see ants moving around us. One of the smallest and most disciplined species found in nature which truly mesmerized us then when we were young and continue to amaze us through their small size which has nothing to do with the huge tasks they accomplish every single day. Ants only strength is in their unity. Unity provides ants with the strength they need to build their nests, strength to save their queen who gives birth to more ants. Ants when working together are disciplined and organized. Their societies have soldiers whose duty is to protect their queen and workers. Workers are regular ants which collects leaves. There is so much we can learn from ants. Ants are important because of their organized way of living and more importantly because of their way of finding the shortest route to their food.

Ants and origin of Co-operation

Building an algorithm inspired by ants was chosen because of its roots of inspiration i.e. ants, and its history. Ants were there before humans and they exist even today definitely proving one simple point that they are one of the oldest and strongest survivors. They survived all kind of weather conditions for millions of years definitely not because they are individually strong but it is because of their innate nature of working together (BBC documentary, 2013). One thing we can learn from the fact is that if creature as simple as ant can exist in climatic conditions varying over millions of years then co-operation is pretty powerful thing. Ants inherit the behaviour of always working in a society and that's what gives origin to the relevance and importance of co-operation.(Yaniv Altshuler, Vladimir Yanovski and Israel A. Wagner,2008) It is not only ants but most species does their task by living in societies and working together co-operatively. If we think of building robots one day as advanced as our self then definitely biology could be a nice place to learn and understand complex events in life that naturally occurs.

Ants seem like a simple creature to imitate computationally and also ants make a good choice because of their innate nature of working co-operatively with other ants. It also makes a good choice because their achievements and potential that cannot be doubted as it has been evident throughout the time. There has been many attempts and many algorithms already exists which

mimics the life of biological inspired species. Ant Colony Optimization (ACO) algorithm is a popular algorithm inspired by ants. Algorithm is often practised through simulation for research purposes in computers where hundreds and thousands of simulated ants work together for finding the route to their food.

Although my algorithms is inspired by ants and undoubtedly ACO helped in understand the behaviour of ants too but my algorithm is different from ACO in many ways.

ACO Algorithm works in the following way. (Leandro Nunes de Castro,2006)

- Ants select different start positions in a given space and starts exploring routes in search of their food as we see in figure 2.2. 1st figure below.
- Once they find food they leave a chemical trace for other ants to follow them called pheromone.
- Various ants will leave traces as we see in figure 2.2 second image below.
- Other ants follow that pheromone depending on the strength of the pheromone.
Strength of the pheromone depends on important factors
 - How many ants travelled that path (Desirability of the route)
 - They also check if there is any other possible unexplored route left to the food.(Attractiveness of the route)
- If there is any unexplored route left then ants will explore the new route and compare to see if the new route is shorter than the old. If new route is shorter than current route's pheromone then current route's pheromone will become stronger and slowly pheromone of older route will evaporate. Else vice-versa will happen as we can see in figure 2.2, third image below.
- Once all the routes are explored then only one pheromone will be left with strongest pheromone, which will also be the shortest route as shown in 4th image in figure below.

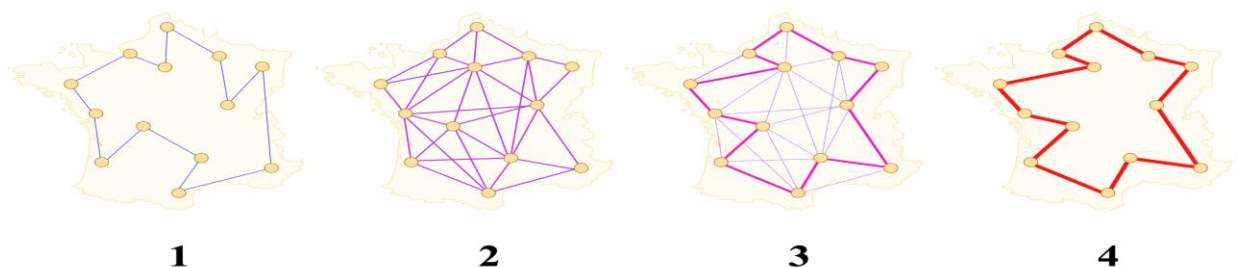


Figure 2-2 ACO

Unlike ACO my algorithm didn't focused on the 2 key factors of ACO algorithm, that is

- Desirability of changing the route. (η_{xy})
- Attractiveness of the route. (τ_{xy})

Where both factors are influenced by the equation below

$$p_{xy}^k = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum_{y \in \text{allowed}_y} (\tau_{xy}^\alpha)(\eta_{xy}^\beta)}$$

Pheromone is measured at any point for any ant k moving from point x to point y by using desirability and attractiveness as influencing factors.

My algorithm is instead based on the probability which proves chances of repeating a single route twice is very rare, when finch is making its movements based on random decisions. I observed this while doing my experiment where same route was never repeated in its 20 recorded iteration period (Appendix 1) and many other unrecorded iterations.

Also ACO cannot be directly implemented in Finch due to the primary requirement of multiple ants working together. Therefore algorithm needs to be modified in many ways to be actually implemented in real world robots, which we'll learn in the working of my algorithm in next chapter.

3 METHODOLOGY

3.1 Introduction

The project is more of an experiment and a research project than a business product but still I believe that the way I did this project was more of an agile approach.

The project was planned and discussed in a group and we used agile approach specifically scrum for the project as we had limited time for completing the project. Almost every day in the morning on our way to the university me and my group partner used to take a tube which takes approximately 30 minutes to reach university, we used to discuss about what we are going to do in those 30 minutes. We also had time to review our work and time to inform each other about the things which will prevent us from compatibility issues. It also helped us stay updated with each other's work. We also used to meet on weekends and test our work which helped me stay on track with the project.

It is important to have some testing method to know if my software properly works. In an experiment to check if my program works as it should, I planned to test the results of the experiment by increasing the number of iterations inside the rectangular walled space. First I will see the distance covered by the finch robot when it ran for 5 times, 10 times, 15 times and 20 times. As number of iterations will increase shortest distance should become shorter and shorter. First run will be counted as the finch robot starting from start position and then stopping at the dark exit box. This process will be repeated till finch have completed its 20 iterations and then program will find the best route from all the data it recorded. This could also help us understand how exactly software works and what it does. We will analyse all the data and see, what is the rate at which finch is improving.

I will also get the data of travelling time from by group partner Sangmok's algorithm in same testing environment and compare the time taken by his algorithm versus my algorithm. I will also try to analyse that under what conditions will my algorithm be better and we will also try to see if both the algorithms could be combined for finding the route more effectively.

If everything will work and data is interpreted as expected then we will conclude that we successfully created a software which is an application of co-operative mobile robotics and we will also succeed in achieving our objectives and aim.

3.2 UML diagrams

UML diagrams are very important for understanding how our system works and they also helps us plan strategies for solving any problem. It is also easier to implement various testing techniques once we have detailed UML diagram of the system. Model Based Testing is a very popular technique and could be used in software system by looking at individual classes or test

cases. Same technique helped me analyse my project in testing period for finding any bugs in the software.

UML diagrams are also used for explaining how program works and implemented. There are 2 diagrams used for planning and explaining the software and project. Two diagrams which I used are:

1. Use case diagram

Every UML diagram has its own significance. Use case diagram is used here to explain the general scenario of how system works. This diagram is a general explanation of how system works for easy understanding. When user will run a file then finch will start exploring random routes and store all the directions in a file. Updating Direction use case will update the directions as explained in table 1. Search best route use case will search for the shortest route travelled by the finch robot from the file containing all the direction data and it will share that data using server use case to client use case using simple client-server program. Client file at any computer will receive the data where analyse data use case will process the data and convert it to compatible format and send it to follow shortest path use case where another Finch waiting for the data will receive the data and start moving from start position to the exit following the shortest route direction just received by it.

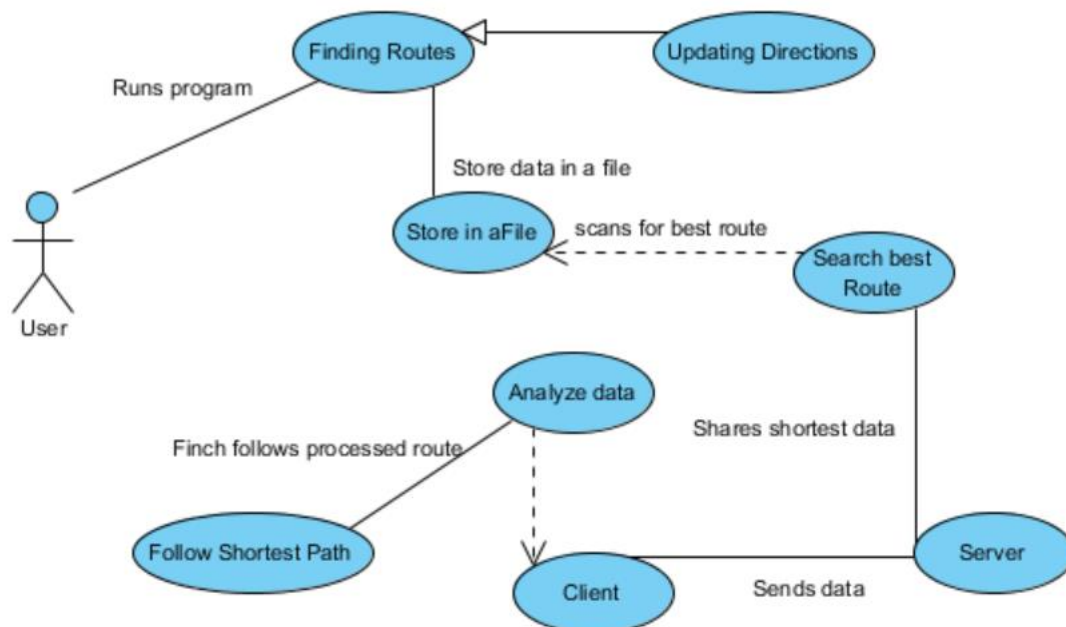


Figure 3-1 Use case

2. Package and class diagram

Below is the package and class diagram of the project. It is a more detailed diagram explaining the actual connection between the classes and the packages. It also lists all the attributes and operators of the class. It is obtained through reverse engineering from the actual program of finding a shortest route. Here ACO is a package which contains 7 classes. Here Function of ACO package is to make finch explore route, collect data and find the shortest route. Class path makes finch explore routes. Class UpdateDirections updates directions from database of directions in table 4 and stores it in a text file. Class ScanningRoutes searches text and converts all the data in compatible format. Class updated pheromone calculates the length of all the distance and finds the shortest distance and also searches for the index where direction is located. This data sent to package server where class finch state reads the data and sends it to class Finch server. Class Finch client sends the data to Second Robo package where data is processed by String Converter and then RoboFollows uses the data from table 1 for interpreting data and sends it to robo2main where everything is executed.

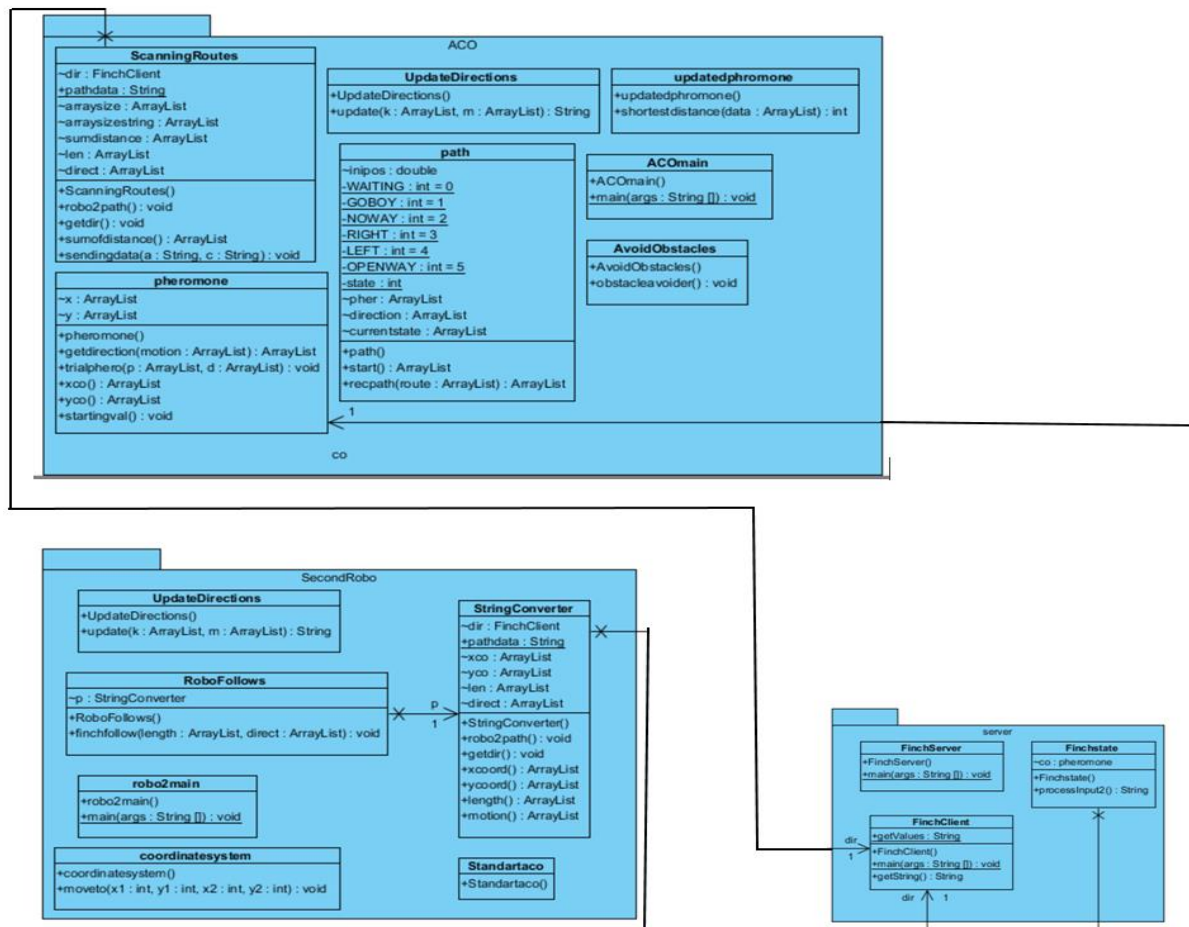


Figure 3-2 Package-class diagram

3.3 Risks

Risks are part of any project and most of the times it is hard to avoid all of them but the best we can do is to have contingencies.

PROBABILITY			
IMPACT		High	Low
	High	<ul style="list-style-type: none">• Finch getting influenced by external forces.• Finch IR sensors not responding.	<ul style="list-style-type: none">• Program crashes
	Low	<ul style="list-style-type: none">• Finch not finding exit.	<ul style="list-style-type: none">• File with direction data deleted/tampered

Table 2 Risk

If we know what problem exists then things become slightly easier as we know where work needs to be done. One of the techniques we will use to achieve our goal is to analyse the problem and then finding our way around the issue.

Contingencies and the way around

Here we have overall 5 issues where 1 issue is at high probability and impact. We will cover all of the issues one by one:

1. Finch getting influenced by external forces.

There is a very high probability that while finch is moving it is influenced by the wire it is connected. The impact will be high too, since the distances are recorded based on time intervals. We can see if wire or any other external force affects the direction of finch then recorded path could be affected with it too since distance is recorded with time and obviously wires could affect finch but not time. We don't want to keep the record of a wrong path. Therefore the best we could do to avoid this problem is to use raspberry pi. Using raspberry pi will not require any wire and with wire gone the issue will be resolved too.

2. Finch IR sensors not responding.

The risk has high probability and high impact. The best we can do for avoiding the problem is by not using any reflecting surface or any long lean objects like leg of chair et al.

3. Finch not finding exit.

It is a high probability low impact risk. Robot might take time but it will eventually find the exit therefore best we can do is either restart program or wait till it finds the exit.

4. File with direction data deleted/tampered

It is low probability and low impact risk. It is very unlikely that the data file is deleted/tampered by itself. We can delete all the data in case data is tampered and run the program again. Finch will start collecting data all over again.

5. Program Crashes

It is again a low probability and high impact risk. It is again unlikely that program crashes by itself. We can either try running the program again or review the program again if problem persists.

3.4 FINCH ALGORITHM

My algorithm could easily be understood using the figure 3.3 below. Finch first travels covering different paths and collecting the data of all the paths as shown in first image of figure 3.1 where different paths are shown by different colors. Later the shortest route, which is denoted by blue colored line, is taken through client server program to another finch waiting at the start position. Now this finch will avoid longer routes and follow the shortest route just received by it.

Here one finch explores all the routes based on the random decisions and it learns from its own history for improving the results. Also in this algorithm finch can send the shortest route found by it to another finch robot without requiring any of the other finches to do any work. This reflects the spirit of co-operating and sharing of data for achieving a common goal where many finches can reach exit covering least amount of distance possible using the same data.

Other finches can also use that data later and continue searching for the improved routes. This also shows the portability of path data which could be used by any finch robot anywhere but it should be noted data will be relevant to the same environment where it was initially tested.

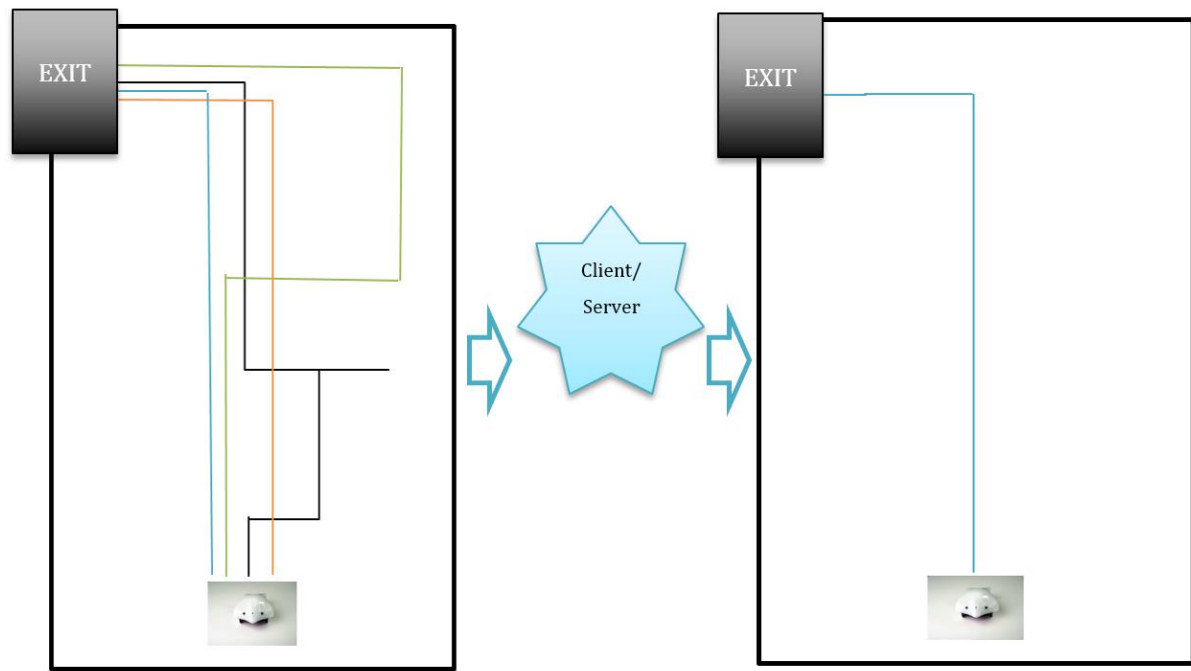


Figure 3-3 Finch Working Model

Co-operative effort by algorithm

Co-operation simply means joint effort where more than one entity work for the common task. My algorithm tries to implement the same quality by using 2 finch robots for achieving the common goal. My program is designed to be a simple application of co-operative mobile robotics. Algorithm assures that finch will find the improved routes with increase in iteration and the portability of data for helping other finch robots achieve the same goal.

4 IMPLIMENTATION

4.1 Introduction

In this chapter I will explain how everything is implemented and how the whole experiment has been performed with the details of the environment set up and details on how finch searches its route. Programming details are covered in chapter 6.

I would like to start by explaining first about the environment in which finch was tested. These details will help us understand the calculation in chapter 7, Evaluation.

4.2 Environment

Finch was tested in a rectangular space of size 100 X 85 cm. Rectangular space size specification could be seen below in figure 4-1. It is rectangular space made up of cardboard and unused boxes. Black box is a square of 25X25 cm square size. Finch was placed at the centre from the bottom as shown on fig 4-1. The first image shows the cardboard-bordered space where finch was tested (table 3, figure 4-2). Finch has been tested in small environment in order to avoid unexpected situations like – uneven surface, reflecting surfaces etc. Second image focuses on the black box which is an exit. It is black because it needs to prevent light from entering inside. Whenever finch detects light lower than its outside environment it stops. The approximate light intensity value which finch received inside the box was 2 while the light intensity outside the black box in the room in regular condition during the period of testing was 67. This is a huge difference and big enough to make finch stop and make it realize that it found the exit.

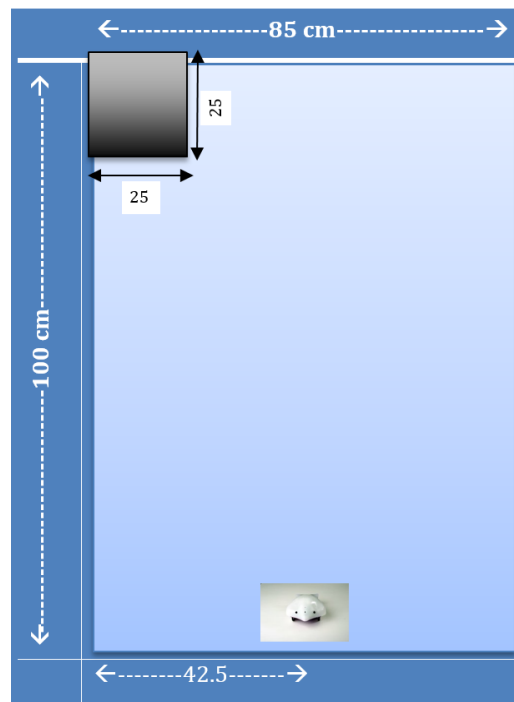


Figure 4-1 Testing space specification



Figure 4-2 Cardboard Bordered space

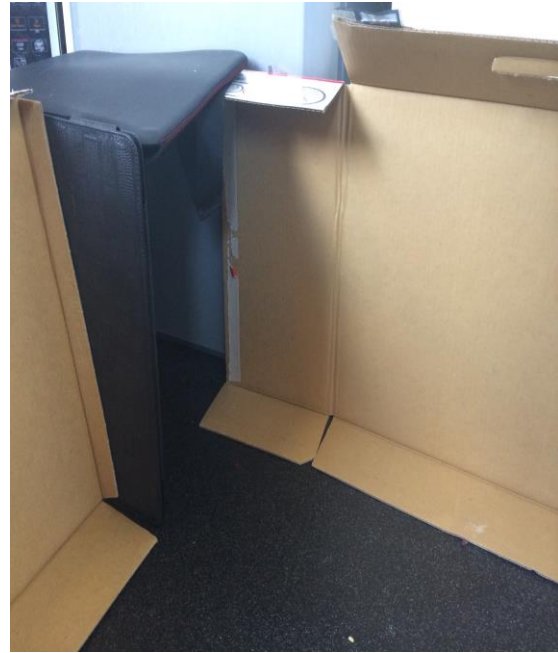


FIGURE 4-3 BLACK BOX

Table 3 Environment images

4.3 Path Planning

It was difficult to keep track of all the directions finch can go due to unavailability of proper path recording hardware in finch. Also finch would have generated huge sets of data to analyse, which might have also affected the compilation of program. I programmed finch to move and record path in only 4 directions namely straight, left, right and back. This will provide simple data set like 20 units straight then 15 units left than something like 15 units at 20 degree left from straight and 10 units at 10 degree from left towards right. Due to time limitation program tries to find the best route only in four directions, which makes it easier to understand. This also results in finch taking 90 degrees turn whenever it wishes to change its direction.

Directions are based on the obstacles detected by sensors and also randomness for exploring different routes. Figure below shows the list of instructions used for keeping track of all the direction of finch in x-y plane. This is also the example of knowledge base explained in figure 2.1 in chapter 2.

Below is database updating the directions of finch as per to our perspective view. Programming implementation of the code is explained in chapter 6 code and working. In order to understand let's consider an example, suppose finch moves straight at starting, maybe because it detects no obstacle, now when it will detect something on right or it randomly chooses to turn left then it will turn left and update the direction array. But after turning left it needs to keep a record that last time it turned left, so now it is moving in left direction. If again next time finch chooses to

move left because it detected something on right then finch needs to know that it is not left from our perspective. It is straight from our perspective because he is already coming from left. Such conditions are needed to keep track of the direction explained below in a table.

FINCH COMING FROM	FINCH FINDS OBSTACLE AT	ACTUAL DIRECTION WHERE FINCH TURNS	UPDATED DIRCTION WHERE FINCH TURNS
Straight	Left	Right	Right
Straight	Right	Left	Left
Straight	Left and Right	Back	Back
Straight	Nil	Nil	Straight
Left	Left	Right	Straight
Left	Right	Left	Back
Left	Left and Right	Back	Right
Left	Nil	Nil	Left
Right	Left	Right	Back
Right	Right	Left	Straight
Right	Left and Right	Back	Left
Right	Nil	Nil	Right
Back	Left	Right	Left
Back	Right	Left	Right
Back	Left and Right	Back	Straight
Back	Nil	Nil	Back

Table 4 Directions Table

4.4 Data Interpretation

Finch starts running randomly for finding the best route as already explained. Finch is programmed to make totally random decisions for finding the exit. It has been done in order to

make finch find different routes. Finch records data of path travelled in a text file, which stays there even when the program is terminated. This means the more number of times finch will explore the space, more chances it will have of finding the better route.

Now we will look at one of the cases from the 20-test iteration, which has been taken in order to explain how finch records its path.

Below is a sample console screen from one of the iterations where finch was searching for the exit. Finch first starts moving straight as it detects no obstacle around it as shown by "Boy is in" in console view1 figure 4-4. Distance is calculated in the program by dividing time taken to complete one move (straight, left, right or back) in milliseconds divided by constant speed 250. Speed is constant of 250 units and time is measured in milliseconds. Time difference in milliseconds is divided by constant value 250 in order to get measurable values. If only milliseconds were used in order to measure the completion time then we were getting very high measuring values or if time is converted to seconds and multiplied by constant speed 250 then values were too small to see the difference. Since speed of finch is not programmed in any known unit and it is relative therefore time difference is divided by 250 constant value in order to get values in same ratio. Dividing the time in milliseconds by constant speed 250 gave nice values which demonstrated visible change in directions. Once program starts it adds straight in an array and then adds the distance covered which are 10.744 units to another array.

Program keeps on adding the directions as it moves. Similarly it also continues to add all the corresponding distances in another array. We can see in the image 4-4 how directions and distances are recorded in 2 different arrays. Program also keeps track of the directions in the form of co-ordinates. Co-ordinates are stored in 2 arrays where first array shown in console is x axis and second array is y axis. We can see first it starts from position (0, 0) and then finch moves to (0, 10). Program keeps on adding new co-ordinates as finch explores new routes.

Below is an image of co-ordinate system, showing how finch moved. All coordinates are stored for keeping track of the direction finch moved. Such explicit tracking of movements also helps in identifying any kind of bugs, which might be causing problems. Co-ordinates could also be verified from the recorded direction and distance arrays.

```
Problems Javadoc Declaration Console
<terminated> ACOmain [Java Application] /Library/Java/JavaVirtual
Connecting to Finch...this may take a few seconds
BOY is in
[Straight]
[10.744]
straight recognized
[0, 0]
[0, 10]
2
2
Going for no left
now
Inside Openway
Going for no way
No way reached
entering update
1
[Straight, Back]
[10.744, 0.224]
Back Recognised
[0, 0]
[0, 10, 10]
3
3
Going for no left
now
Inside Openway
Leaving Now
Going for no left
now
Inside Openway
Going for no way
No way reached
entering update
2
[Straight, Back, Straight]
[10.744, 0.224, 0.224]
straight recognized
[0, 0, 0]
[0, 10, 10, 10]
```

Figure 4-4 console View 1

```
Going for no left
now
Inside Openway
Leaving Now
Left reached
3
[Straight, Back, Straight, Left]
[10.744, 0.224, 0.224, 2.912]
Left Recognised
[0, 0, 0, 0, -2]
[0, 10, 10, 10, 10]
5
5
```

Figure 4-5- Console View 2

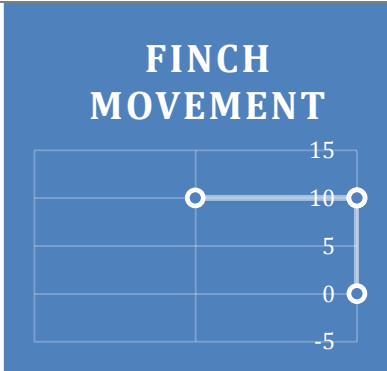


Figure 4-6 Finch Movement in co-ordinate System.

Table 5 Console views and co-ordinate system

This process is repeated many times in order to get more and more data for finding the better route. This process is repeated 5 times, 10 times, 15 times and 20 times and compared to see if finch succeeds in finding the shorter route. We will focus on programming perspective of the software in chapter 6.

5 TEAM EFFORT

This chapter focuses on my group partner Sangmok's and my collaborative effort for researching on the common area of co-operative mobile robotics. We started working in a same field and attempted to study the benefits of co-operative mobile robotics but our approach for achieving the same goal was different. I worked on biology and ants for inspiration while my group partner worked on his own algorithm in which finch follows the light from anywhere and stops once it reaches the light source. First we will try to understand and see how this algorithm works and then we will try comparing and if feasible then combining the algorithms.

5.1 Follow Light Algorithm

This algorithm doesn't only follow light but it follows any light intensity it is asked to follow. For example anyone goes to the place where he wants finch to go and records the intensity of light at that place and feeds the values to the program. Program will direct finch towards that light intensity. This also means that finch will not only follow brightness but it could also follow any darkness too.

5.2 Limitations

Some of the limitations of the algorithm are:

- It would be hard for finch to find places where light intensity is even everywhere.
- This algorithm could be used only when the intensity of light is known.

5.3 Combing Algorithms

My group partner's algorithm focusses on the use of light source for finding the exit. It follows the light source pretty accurately. Light follow algorithms if combined with my algorithm and implemented and tested in the same environment which I implemented earlier in figure 4-2 but with the exception of the black box. Instead of black box I'll be using lamp which will help informing finch about its way to the exit as shown in figure 5-1 below. This new algorithm will probably be more accurate and faster for finding routes if combined with my algorithm.

Whole idea of using light source at the exit is that since there is no GPS devise installed inside the finch, it is hard to tell finch where it needs to go. Therefore light could be used as GPS location for guiding where finch needs to go. Similarly in my project too black box is used to make finch realize that it found the exit while in practical situations co-ordinates of the location where we want finch to go and search for the best routes be used.

Currently Finch moves randomly with my algorithm which may cause it to take wrong decisions or it may even take many circular rounds in the same space before finding the exit. Therefore if

finch could see the light source (or knows the location where it needs to go) at the exit and follows it and now when it is iterated for 20 times, we will have 20 pretty nice routes where each route is pretty good on itself since in every iteration finch was aware where needs to go. Earlier in my algorithm finch didn't know where it is going until it reached its destination.

We will test how much time does this algorithm takes for finding the exit by iterating the program ten times. We will then compare and analyze the range of time variation which finch takes for finding the light source at the exit with the time variation in my algorithm. We will also see how both the algorithms could be merged in chapter 7.



Figure 5-1 Light Follow

6 CODE AND WORKING

This chapter focuses on programming and technical aspect of the project. Algorithm is programmed in java using eclipse junio.

Path planning

Path planning has been a crucial part of programming. Finch uses switch cases for switching from 6 cases namely LEFT, RIGHT, NOWAY, GOBOY, OPENWAY, and WAITING which while explaining I'll refer as states. Program starts by going to state WAITING where it checks for obstacle. If program detects any obstacle it will go to state OPENWAY where it finds position of the obstacle. If obstacle is at right it will turn left else vice-versa. If anytime at state WAITING finch detects no obstacle then there is 50 percent probability that finch will either go left or right and 50 percent probability that it will go straight. This gives us 25% probability of going left and 25% probability of right (code snippet 6-1). This makes sure that finch don't only makes turns when it detects obstacle but otherwise also. This way program keeps of switching from one state to another until stopping condition is met.

Below is the snippet of code displaying how finch makes random decision for exploring routes.

```
if(myFinch.isObstacle()==true){
    state=OPENWAY;
    System.out.println("Going for no left");
}

else{
    Random r2 = new Random();
    int coin = r2.nextInt(2);
    if(coin==0){
        int coin1 = r2.nextInt(2);
        if(coin1==0){
            state=LEFT;
        }
        else{
            state=RIGHT;
        }
    }
    else{
        state=GOBOY;
        System.out.println("boy is moving straight");
    }
}

System.out.println("now");
break;
```

Figure 6-1 Random movement code

The code snippet is inside the case WAITING (starting of code at Figure 6-4).

This explains how finch travels but recording and updating its path as explained in table 4 is next important task.

```
case LEFT:
    if(myFinch.getLeftLightSensor()<5 && myFinch.getRightLightSensor()<5){state= WAITING;break;}
    currentstate.add("LEFT");
    //for(int i=0;i<myFinch.getLightSensors().length;i++){
    //}
    double nrs=System.currentTimeMillis();
    System.out.println("Left reached");
    // myFinch.setLED(0,0,255);
    myFinch.setWheelVelocities(-250,-250, 200);
    myFinch.setWheelVelocities(-150, 150, 500);
    double nre=System.currentTimeMillis();
    double dis1 = (nre-nrs)/250;
    direction.add(update.update(direction,currentstate));
    pher.add(dis1);
    hello.getdirection(direction);
    recpath(pher);
    hello.trialphero(pher,direction);
    if(myFinch.getLeftLightSensor()<5 && myFinch.getRightLightSensor()<5){state= WAITING;break;}
    state=WAITING;
```

Figure 6-2 LEFT CASE code

In code snippet 6-2 we can see the code

"DIRECTION.ADD(UPDATE.UPDATE(DIRECTION.CURRENTSTATE));"

This code adds the updated direction where update is an object of another class called UpdateDirections used for initializing a method called "update". Method uses 2 arrays as parameters where "direction" tells the method where finch is coming from and "currentstate" tells program where finch is intending to go.

```
public String update(ArrayList<String>k,ArrayList<String>m){
    String state="null";
    System.out.println(k.size());
    if(k.get(k.size()-1).equals("Straight")){
        if(m.get(m.size()-1).equals("RIGHT")){
            state = "Right";
        }
        else if(m.get(m.size()-1).equals("LEFT")){
            state="Left";
        }
        else if(m.get(m.size()-1).equals("NOWAY")){
            state="Back";
        }
        else{
            state="Straight";
        }
    }
}
```

Figure 6-3 updating direction code

Code in figure 6-3 explains the code from update method. Code says if finch is coming from straight and chooses to go right then add, "Right" in the direction array. Similarly if finch is coming from left, right or back then data is updated as referred in Table-4 by following the same coding style.

In figure 6-2 we can see that variable "nrs" records the time at the starting when finch starts turning left and variable "nre" records time at the end of the movement of finch. Difference of the value is divided by constant value 250 to get a measurable value as explained in section 4.4. This is the new distance and added to array caller "pher". This distance will correspond to the updated direction.

Black Box and Path data

Next important step is to have a stopping condition and to have ability to record travel data somewhere secure. Stopping condition is met by using "if" condition which makes sure if sensor detect any value lower than 5 then finch will stop.

```
if(myFinch.getLeftLightSensor()<5 && myFinch.getRightLightSensor()<5)
```

As we have seen while finch is moving it is updating and recording the travel path in an array. Program writes all values from array in a database file called "pherall.txt" using the code snippet in figure 6-4. Program uses "," to separate directions and last direction value is ended with ".", so later while reading values, program knows it reached the end of characters. Similarly

distances are also separated by “,” but ended with “+” since distances are decimal values and they already have lot of dots therefore another dot cannot be used for making program stop reading the values. Also program adds “/” in order to keep direction and distance values separate.

```
case WAITING:
    if(myFinch.getLeftLightSensor()<5 && myFinch.getRightLightSensor()<5){
        //if(myFinch.isBeakDown()==true){
        try(PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter("pheroall.txt", true)))) {
            for(int i=0;i<direction.size();i++){
                if(i==direction.size()-1){
                    out.write(direction.get(i)+ ".");
                }
                else{
                    out.write(direction.get(i) + "," );
                }
            }
            out.write("/");
            for(int i=0;i<pher.size();i++){
                if(i==pher.size()-1){
                    out.write(pher.get(i)+ "+");
                }
                else{
                    out.write(pher.get(i) + "," );
                }
            }
        }
    }
```

Figure 6-4 writing path code

Data is stored as explained in section 4.4. File “pherall.txt” which could be found in APPENDIX B1, Dataset.

Next we need program to read data from file “pherall.txt”, and search for the shortest route covered by finch from database.

ScanningRoute class does the part of reading all the characters from the text file and converting it back to double and string values.

This code reads all the data from the text file.

```
public void getdir() throws IOException{
    updatedphromone a = new updatedphromone();
    BufferedReader br = new BufferedReader(new FileReader("pheroall.txt"));
    try {
        StringBuilder sb = new StringBuilder();
        String line = br.readLine();

        while (line != null) {
            sb.append(line);
            sb.append(System.lineSeparator());
            line = br.readLine();
        }
    }
```

Figure 6-5 Reading path code

Next read data is stored in string “z”. Now it is required to separate the recorded distance with the direction because direction needs to be stored as string arrays while distance needs to be stored as double array.

```
System.out.println(pathdata);
String z=pathdata;
String[] k = {};
int count=0;
for(int l=0;l<z.length();l++){
    if(z.charAt(l)=='/'){
        count++;
    }
}
System.out.println(count);
k=z.split("/",count);
System.out.println(z);
for(int i=0;i<k.length;i++){
    System.out.println("This is a string array"+i +" "+k[i]);
}
```

Figure 6-6 Reading splitting path code

Split condition splits all the stored data at slashes (“/”) and stores the values in string array k. Since direction is stored first and distance later and from the code above it could be seen that all the directions and distances will be stored inside array k. Since we split the data using split command therefore at array k’s even index values, directions will be stored and at odd index values, all the distances values will be stored.

Now all we need is the sum of all the distances and find the smallest distance in the database. We now already know that distance is stored in array k odd indexes places. Now all we need is to covert values at arrays odd index places to double values and store their sum at another array for finding the shortest route.

```
else{
    System.out.println(n);
    for(int j=0;j<=k[n].toCharArray().length;j++){
        //System.out.println(o.length + "equals"+j);
        String temp="";
        while(k[n].charAt(j)!=''){
            if(k[n].charAt(j)=='+'){break;}
            temp+=k[n].charAt(j);
            j++;
        }
        double b=Double.parseDouble(temp);
        len.add(b);
        if(k[n].charAt(j)=='+'){break;}
    }
    int u=len.size();
    arraysize.add(u);
    System.out.println(arraysize);
    if(n>1){
        for(int j=0;j<arraysize.get(arraysize.size()-2);j++){
            len.remove(0);
        }
        arraysize.set(arraysize.size()-1, len.size());
    }
    System.out.println("This is the distace"+len);
    double sum=0;
    for(int i=0;i<len.size();i++){
        sum+=len.get(i);
    }
    sumdistance.add(sum);
    System.out.println("This is the sum"+sum);
}
n++;
}
a.shortestdistance(sumdistance);
sendingdata(k[(2*a.shortestdistance(sumdistance))+1],k[2*a.shortestdistance(sumdistance)]);
```

Figure 6-7 Scanning Route code

This code helps us convert values to double and finding the sum of distances. Code keeps track that it only gets odd values from array k for converting them to double and then adds it in array list “len”. Array List arraysize is used for keeping track that “sum” array doesn’t reach out of bound condition. Finally sum of all the distances is calculated and stored in a variable “sum” as shown in code snippet above. This sum is every time added to an array list “sumdistance” which keeps track of sum of all the distances in each run.

Now another class “updatedphromone” finds the index containing the smallest value. (Appendix B, B2)

Scanning Route class finds the value of index containing smallest value using object “a” which initializes the method “shortestdistance” from class “updatedphromone” as shown on screenshot below. This index when multiplied by 2 and incremented by 1 gave the position of shortest distance in array k and when index is just multiplied by 2 it gave the position of shortest direction in array k.

```
        a.shortestdistance(sumdistance);
        sendingdata(k[(2*a.shortestdistance(sumdistance))+1],k[2*a.shortestdistance(sumdistance)]);
    }
    public ArrayList<Double> sumofdistance(){
        System.out.println("This is total distance covered"+sumdistance);
        return sumdistance;
    }
    public void sendingdata(String a,String c){
        try {
            BufferedWriter out = new BufferedWriter(new FileWriter("shortestdistance.txt"));

            out.write(c);
            out.write("/");
            out.write(a);

            out.close();
        } catch (IOException e) {}
    }
}
```

Figure 6-8 Writing shortest distance code.

We can see in the screenshot that finally this shortest distance directions are stored in a text file called "shortestdistance.txt" (Figure 7.3).

Once shortest distance is found next task is to send this shortest distance to another finch using client server program.

Server program used class Finchstate in order to read the shortest distance and then at port 4444 it sends the shortest route to the receiving client class.

```
ServerSocket serverSocket = null;
try {
    serverSocket = new ServerSocket(4444);
} catch (IOException e) {
    System.err.println("Could not listen on port: 4444.");
    System.exit(1);
}
System.out.println("Server up and waiting");
```

Figure 6-9 Server connection code

Screen shot at figure 6-10 of client server program shows how server-client connection at socket port 4444 is established. Once the connection is established then shortest route data is written in host computer's hard drive by the name of "Clientfile1.txt".

```
try {
    medSocket = new Socket("localhost", 4444);
    out = new PrintWriter(medSocket.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(medSocket.getInputStream()));
} catch (UnknownHostException e) {
    System.err.println("Don't know about host: localhost ");
    System.exit(1);
} catch (IOException e) {
    System.err.println("Couldn't get I/O for the connection to: 4444.");
    System.exit(1);
}

BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
String fromServer;
System.out.println("Initialised client and IO connections");
while ((fromServer = in.readLine()) != null) {
    System.out.println("Server: " + fromServer);
    getValues+=fromServer;
    try {
        BufferedWriter out1 = new BufferedWriter(new FileWriter("Clientfile1.txt"));

        out1.write(getValues + " ");
        // out1.write(getValues + " ");

        out1.close();
    } catch (IOException e) {}

    if (fromServer.equals("Bye."))
        {break;}
}
```

Figure 6-10 Client code

Now String Converter class in another package named SecondRobo will read the file written by client class and convert the data received back to strings and double values in a similar way as it did earlier (Figure 6-7). Now update directions class similar to update direction class in ACO package will check where finch is coming from and where it wishes to go in order to follow the received path.

Code below at figure 6-11 first converts direction and distance values in appropriate form and then RoboFollow class's method finchfollow is initialized which takes distance and direction as parameters in order to make finch follow the shortest route. Finchfollow is programmed in similar fashion as shown in figure 6-3.

```
public static void main(String[] args) throws IOException {  
    // TODO Auto-generated method stub  
    StringConverter z= new StringConverter();  
    RoboFollows my = new RoboFollows();  
    z.getdir();  
    my.finchfollow(z.length(), z.motion());  
}
```

Figure 6-11 Second Finch main class

Program keeps on recording the data for finding the best path. Algorithm could be used even after many days and finch will continue searching for the better route than the last route it found many days ago as long as data is secured and environment is same..

7 EVALUATION

7.1 Introduction

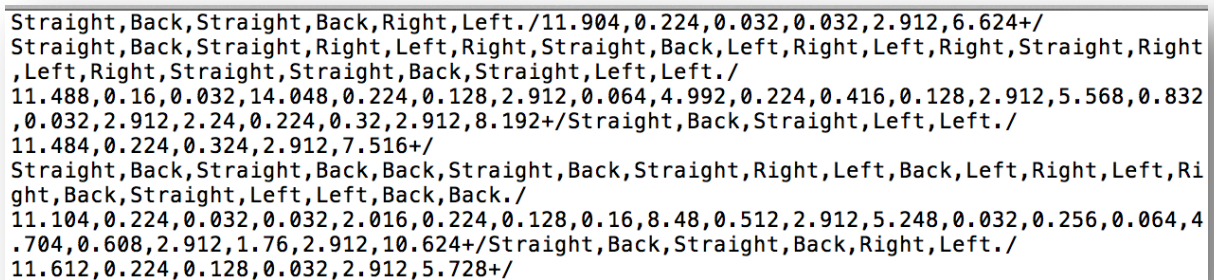
In this chapter we will be testing and evaluating the algorithm and study the results and compare them for getting the better picture of the whole dissertation. I have tested the program by running finch 5 times, 10 times, 15 times and 20 times. Now we will do our experiment. Here we start our experiment with 5 iterations. Raw data of the directions recorded in all 20 iterations is given an Appendix B.

7.2 Algorithm Testing

Here we will analyse all the results obtained up to 5th, 10th, 15th and 20th iteration.

- 5 iterations

Below is image showing the recorded log of all the distances travelled by finch robot while finding the exit. It shows log of 5 iterations. Here directions are separated using “/” with the distance it travelled. Directions directly correspond with the distance it travelled. For example in first iteration finch moved straight for 11.904 units.



```
Straight,Back,Straight,Back,Right,Left./11.904,0.224,0.032,0.032,2.912,6.624+/  
Straight,Back,Straight,Right,Left,Right,Straight,Back,Left,Right,Left,Right,Straight,Right  
,Left,Right,Straight,Straight,Back,Straight,Left,Left./  
11.488,0.16,0.032,14.048,0.224,0.128,2.912,0.064,4.992,0.224,0.416,0.128,2.912,5.568,0.832  
,0.032,2.912,2.24,0.224,0.32,2.912,8.192+/  
Straight,Back,Straight,Left,Left./  
11.484,0.224,0.324,2.912,7.516+/  
Straight,Back,Straight,Back,Back,Straight,Back,Straight,Right,Left,Back,Left,Right,Left,Ri  
ght,Back,Straight,Left,Left,Back,Back./  
11.104,0.224,0.032,0.032,2.016,0.224,0.128,0.16,8.48,0.512,2.912,5.248,0.032,0.256,0.064,4  
.704,0.608,2.912,1.76,2.912,10.624+/  
Straight,Back,Straight,Back,Right,Left./  
11.612,0.224,0.128,0.032,2.912,5.728+/  

```

Figure 7-1 Distance and directions file

Program scans all the data below in screenshot image 7-2. Program sums up the length of the distances recorded for finding the shortest path as explained in chapter 6 (figure 6-7). Next program compares the length of distance with all the other distances for finding the shortest route.

Program stores the sum of the distances recorded in each iteration by finch in an array “sumdistance” (figure 6.7). We can see in figure 7-2 that it recorded the sum of distance in first iteration as 21.727, then 60.9599 for second, 22.46 for third, 54.94399 for fourth and 20.636 for fifth iteration. Here we can clearly see that 4th index contains the smallest value, which is 20.636 in this case. It should be kept in mind that counting of an index of an array starts from 0. We can see in the screenshot that it says “index is: “4” for the shortest distance. This index is

used for identifying the location of the shortest distance from all the distance stored in a file whose screenshot is displayed in image 7.1

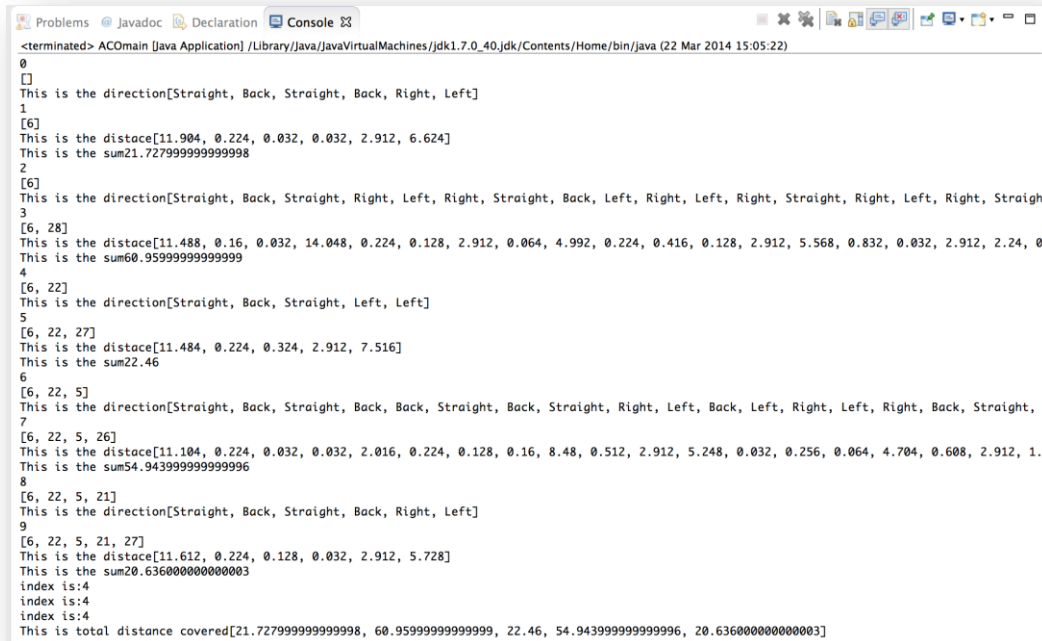


Figure 7-2 console view of searching shortest route

This distance is then taken and stored in another file whose screen shot could be seen below in figure 7-3. This is the distance, which is sent to another robot for following the shortest route.

Shortest Distance

[illegible]

Figure 7-3 Shortest Distance at 5th iteration

Here when we performed 5 iterations we can see that the shortest distance was found at index 4 or 5th iteration, which is clearly displayed above, and the sum of all the distance covered by finch at its fifth iteration is 20.636.

- Iterations 10

When we iterated Finch for 10 times, we witnessed no improvement and still the best route remained same at index 4 or the 5th iteration.


```
index is:4
This is total distance covered[21.727999999999998, 60.95999999999999, 22.46, 54.943999999999996,
20.636000000000003, 28.671999999999997, 23.836, 22.784, 27.775999999999996, 25.312]
50°E3E0000000000003' 58°E170000000000001' 53°830' 55°184' 51°112000000000000' 52°375]
```

Figure 7-4 Sum of distance upto 10 iteration

Therefore the best route found by finch is still same as it was earlier at 5th iterations. We could see the shortest distance at figure 7-3.

Iterations 15

Now when we ran finch for 15th time, we saw improvement and finch succeeded at finding a better route at 14th iteration.

```
index is:13
This is total distance covered[21.727999999999998, 60.95999999999999, 22.46, 54.943999999999996, 20.636000000000003, 28.671999999999997,
23.836, 22.784, 27.775999999999996, 25.312, 66.875999999
45.983999999999995, 20.156, 20.128, 58.587999999999994]
```

Figure 7-5 Sum of distance a upto 15 iteration

At 13th index we can see that sum of the distances covered by finch is 20.128, which is slightly lower than 20.636, which was earlier the shortest distance. Now the pheromone is updated and the new shortest distance is shown in a screenshot below at figure 7-6.

```
Straight,Back,Straight,Back,Back,Straight,Back,Straight,Left,Left./
11.232,0.16,0.032,0.032,1.856,0.224,0.032,0.224,2.912,3.424+/'
```

Figure 7-6 shortest distance at 14th iteration

This data is stored in a separate text file called “shortestdistance.txt”, which is now ready to be sent to another finch.

- Iteration 20

When finch ran for 20 times in again managed to find a better route at 19th iteration as we can see from the screenshot below at figure 7-7. Sum of all the distance covered by finch at its 19th iteration is 16.128, which is lower than its earlier shortest distance, which was 20.128.

```
index is:18
This is total distance covered[21.727999999999998, 60.959999999999999, 22.46, 54.943999999999996, 20.636000000000003, 28.671999999999997,
23.836, 22.784, 27.775999999999996, 25.312, 66.875999999
45.983999999999995, 20.156, 20.128, 58.587999999999994, 130.78000000000003, 22.56, 23.256, 16.128, 60.435999999999999]
```

Figure 7-7 Sum of distance a upto 20 iteration

Below at figure 7-8 we can see the shortest distance found by finch in the duration of 20 iterations.

Shortest distance is

```
Straight,Back,Straight,Back,Right,Left./10.208,0.032,0.16,0.256,2.912,2.56+
```

Figure 7-8 shortest distance at 19th iteration

7.3 RESULT ANALYSIS

Below is a chart showing the best routes at various iterations in my algorithm. Below we can clearly see how route selection improved over time with increase in iterations. Blue line shows the best route covered by finch till its 5th and 10th iteration. Best route found till 15th iteration is showed by the red line, which enters black box and covers slightly more distance than the best route found by finch at 19th iteration, which is showed using green line. Green line here is the new and final pheromone which other finches can follow.

These directions helps us understand how finch moves, also provides us data for analysing and improvising the movements of finch.

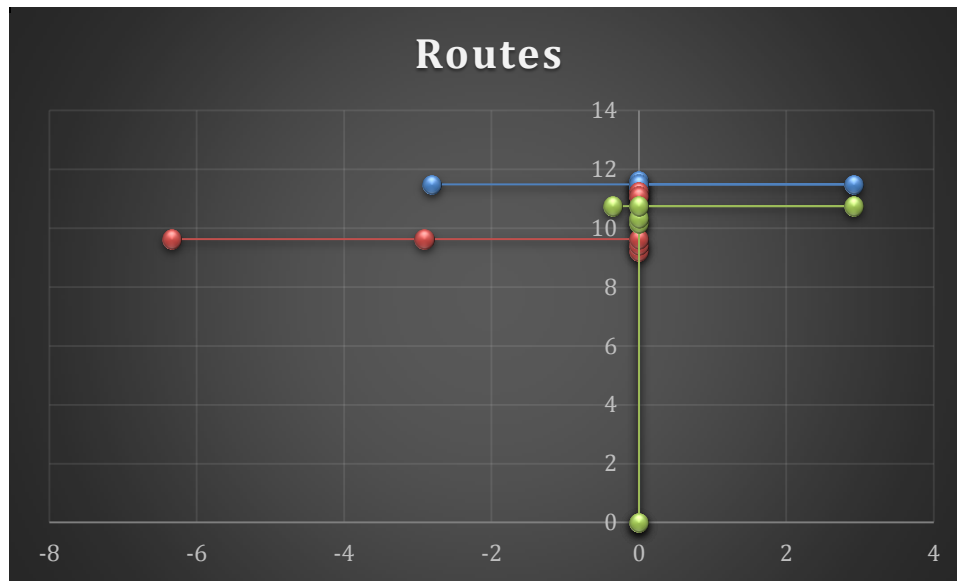
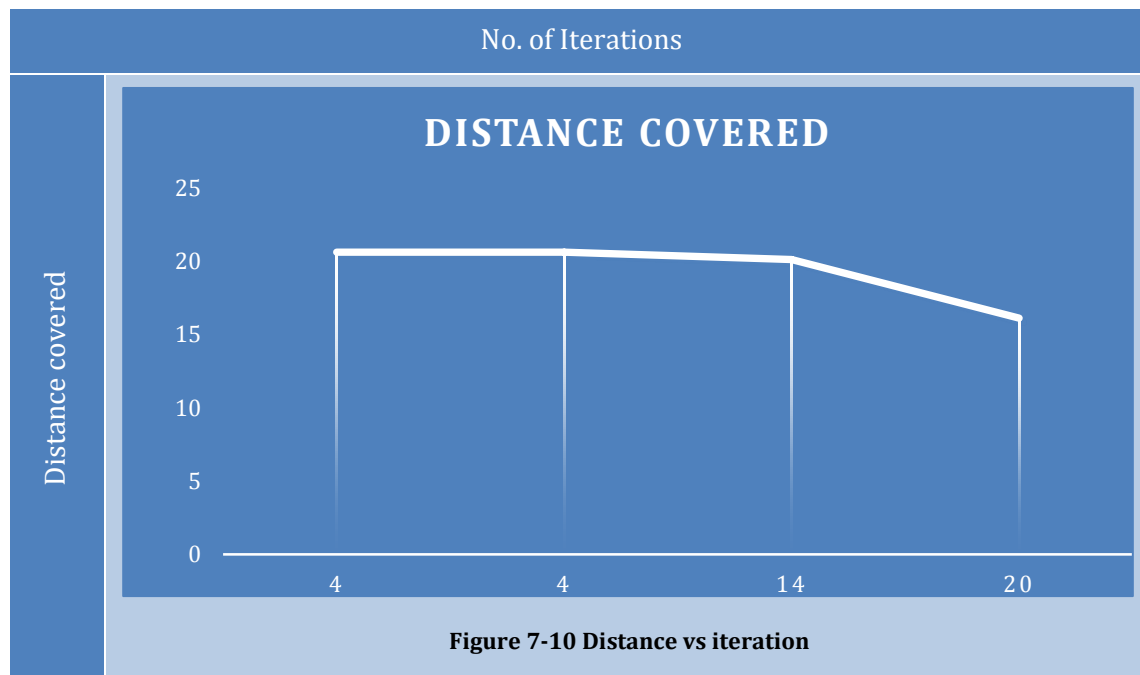


Figure 7-9 Directions

Overall there has been reduction in length of the distance covered by finch. We have seen from the results that up to tenth iteration, the best and shortest route found by finch was 20.636 units. We saw slight improvement at 14th iteration when the overall distance covered by finch was 20.128 units. Again at 19th iteration we saw some significant improvement in the distance, which again reduced from 20.128 to 16.128 units. Below is the graph in figure 7-10 showing how distance is reducing with increase in the number of iteration.

This experiment showed us that with increase in numbers of iterations we have more chances of finding the better route. Since searching for the route is random there could also be the possibility where finch finds the best route in its first iteration, but probability of getting the best route is very low considering the existence of thousands of possible route to the exit. Clearly we have achieved first three objectives for achieving our aim and also since best route is found finch can now easily send the new shortest route distance to another finch using client server program as explained in chapter 6.

Now we have successfully succeeded in completing first 4 objectives. Now we will head for our final objective of studying and analysing the feasibility of combination of 2 algorithms.



7.4 EVALUATING & TESTING GROUP WORK

Here I have tested my group partner's algorithm in my environment where I have used light source at the exit instead of the black box as shown in figure 5-1 and I received following readings in 10 iterations.

Iteration	Time (in seconds)	Movements
1	17.868	28
2	27.503	37
3	10.26	20
4	11.971	23
5	7.409	15
6	6.649	17
7	17.93	27
8	9.682	19
9	9.765	18
10	7.418	15

Table 6 Light algorithm time and movements

This algorithm don't depends on iteration and neither it improves over time but if combined with my algorithm we will see that up to 5 iteration best distance covered by algorithm is at 5th iteration which is 7.409 seconds and up to 10 iteration the shortest distance covered will be at 6th iteration with 6.649 seconds (Table 6).

Finch ran 10 times in the testing environment and the time taken in each iteration is given in time section of the table 5. On average finch took 12.6455 seconds to reach exit in follow light algorithm.

Shortest time taken by this algorithm is 6.649 seconds at 6th iteration.

Table below shows my best results which we obtained by running finch 20 times in section 6.1 converted in seconds. Since distance covered is calculated in program by dividing 250 to the time in milliseconds therefore we can get time in seconds by

$$\text{Time (in seconds)} = (\text{distance} * 250) / 1000$$

Iteration	Time	Movements
Up to 5 th Iteration	5.159	6
Up to 15 th Iteration	5.032	10
Up to 20 th Iteration	4.032	6

Table 7 best time and movements

For example 20.636 which was the shortest distance until 5th iteration will be converted to seconds using

$$\text{Time (is seconds)} = (20.636 * 250) / 1000 = 5.159 \text{ seconds}$$

In my algorithm overall in 20 iteration there was average route timing of 9.673 seconds and in 10 iteration average route timing of 7.725 seconds. In both the cases average time taken to find exit in my algorithm is lesser that 12.645 seconds, which is the average time for finding the exit in follow light algorithm.

Although it could not be guaranteed that every time average time to find the exit in my algorithm will be lower than follow light algorithm. But chances are likely in shorter distances.

Iteration	Time (in Seconds)	Iteration	Time (in seconds)
1	5.43	11	16.718
2	15.23	12	11.495
3	5.615	13	5.029
4	13.735	14	5.032
5	5.159	15	14.646
6	7.167	16	32.695
7	5.959	17	5.64
8	5.696	18	5.814
9	6.94	19	4.031
10	6.328	20	15.1

Table 8 time taken in 20 iteration

Interpreting results

	Light follow Algorithm (Sec)	My algorithm (Sec)
Average Time(10 iteration)	12.6455	7.725
Average Time (20 iteration)	Not Tested	9.673
Least time up to 5 th iteration	7.409	5.159
Least time up to 10 th iteration	6.649	5.159
Overall least time	6.649	4.032
Variation	6.649 to 17.868	4.032 to 32.695

Table 9 Result Summary

Overall my algorithm succeeded in finding a better route at lesser time but in light algorithm time taken by finch to find exit ranged from 6.649 to 17.868 seconds (Table 5) while in my algorithm finch time ranged from 4.032 to 32.695 seconds (Table 7). This reflects variety of routes in my algorithm. Result also shows that my algorithm has lot of very long ways too as finch is not aware of the exit till it finds it (Appendix B1). If light algorithm is combined with my algorithm then finch could know where it is going and that could help it reduce its search area.

We can say that if finch knows where it needs to go then finding shortest route over longer distances could be easier than checking all possible routes. As I also explained earlier that light source is just analogous to GPS guiding finch where it needs to go.

Although if finch's position cannot be determined or we don't exactly know where finch exactly needs to go in cases like space explorations searching for water traces then random route search kind of exploration algorithm can be used for finding the best route to the desired destination.

It could be said that in longer distances combination of both the algorithm could be found useful. In the given testing environment my algorithm seems to have found the shorter route without the need of light source proving effective for shorter distances.

Although combining algorithm is feasible and could be done for longer distance. This analysis also completes my final and 5th objective and successfully achieves its aim.

8 CONCLUSIONS

It is indeed true that shortest path planning through an algorithm inspired by ants is a very small step in the field of Co-operative mobile robotics but one step at a time can help us cover a huge distance one day.

This project is an attempt at implementing an application of co-operative mobile robotics and discussing its benefits and limitations. This application demonstrated how robots can work together making things and complex tasks easier for our society. In this project finch robots with some basic hardware were used which proves that implementing algorithm might be easier and cheaper. Project and developed successfully completed all our specified objectives resulting in achieving our aim. Currently such algorithms and application might be only used in controlled testing environment as they might not be ready yet to be launched in real world. If people will use the algorithm for applications like foraging, finding shortest route or any other purpose in real world these robots might need to be upgraded for better understanding of the geometry of uneven surfaces. There are many industries which are working on developing advanced robots like Boston dynamics. These companies already achieved making advanced robots which can walk on uneven surfaces like roads, desserts, rain and snow. Although most of them are still in research phase (Samuel Gibbs, 2013). If such applications could be implemented easily on all the robots then they might be able to work co-operatively in a real world environment until then they need to work and evolve in testing and controlled environment.

We also learned through short path finding application about how some simple robots like finch be used for demonstrating the technology in the area of co-operative mobile robotics. If enough time, resources and enormous available data are used then undoubtedly much more could be done in the area for the betterment of our society (Dinham, M, Gu Fang, 2007).

Today in our world the field of robotics is slowly making its own space in the industry. Here co-operative mobile robotics is at attempt of seeing the world of future where robots and other artificial entities can communicate among themselves and take simple decisions by themselves.

8.1 Future Work

There are many things which could be done in the field and in this project. There are some improvement I am looking forward to do in this experiment mainly includes some hardware and some software improvement. Hardware improvement might include using advanced obstacle detecting sensors like sonar or laser which could not only verifies the existence of the obstacle but also detects the distance at which obstacle is placed. This could help finch get better awareness of its environment. Also I would like to test the same experiment using GPS technologies which will help us get to know the exact location of the finch, which will also be

helpful for finding the exact distance covered by the finch. Instead of using a black box to stop finch when it reaches its destination, GPS could be used for providing the exact location where we want to send finch.

At software end there are many things which couldn't be completed due to time limitation but I would like to program more than one finch for finding the shortest route together. Also program them to detect each other's presence and avoid any kind of collisions. I would also like to program finch to detect and go in any direction and not only straight, left, right and back. All this changes could also help us understand the ramifications which we should know before implementing the technology in the world.

I would also like to work in this field with my group partner and try to combine his algorithm with mine which could be really helpful in situations where finch could find the exit or any place based on the light sensitivity. If we tell finch about what is the approximate light intensity where it needs to go then it will not only find that place but also try to find the best route to that place. Data management is very crucial for improving. Program learns from its mistakes and its history therefore its history and recorded data path needs to be securely placed and more data mining techniques should be implemented for improved results and smarter decisions.

References

- LEANDRO NUNES DE CASTRO, (2006) "SWARM INTELLIGENCE", FUNDAMENTALS OF NATURAL COMPUTING. CHAPMAN & HALL/CRC, PP 205-256.
- MICHAEL NEGNEVITSKY, (2005) ARTIFICIAL INTELLIGENCE A GUIDE TO INTELLIGENT SYSTEMS.
- GU FANG, GAMINI D., HAYE LAU (2004) 'A BEHAVIOR-BASED OPTIMIZATION STRATEGY FOR MULTI-ROBOT EXPLORATION', CONFERENCE ON ROBOTICS, AUTOMATION AND MECHATRONICS, VOL-2, PP 875-879. [ONLINE] DOI: 10.1109/RAMECH.2004.1438033.
- DINHAM, M.; GU FANG (2007) "TIME OPTIMAL PATH PLANNING FOR MOBILE ROBOTS IN DYNAMIC ENVIRONMENTS", MECHATRONICS AND AUTOMATION. ICMA 2007. INTERNATIONAL CONFERENCE ON, ON PAGE(S): 2132 – 2137.
- MASUDA, M., WEHNER, N. , XIAO-HUA YU (JUNE, 2010) "ANT COLONY OPTIMIZATION ALGORITHM FOR ROBOT PATH PLANNING" , COMPUTER DESIGN AND APPLICATIONS (ICCD), 2010 INTERNATIONAL CONFERENCE ON (VOLUME:3), PP 436-439. [ONLINE] DOI: 10.1109/ICCD.2010.5541300
- FUKUNAGA, A.S. ; KAHNG, A.B. ; MENG, F.(1995) "COOPERATIVE MOBILE ROBOTICS: ANTECEDENTS AND DIRECTIONS", INTELLIGENT ROBOTS AND SYSTEMS 95. 'HUMAN ROBOT INTERACTION AND COOPERATIVE ROBOTS', PROCEEDINGS. 1995 IEEE/RSJ INTERNATIONAL CONFERENCE ON (VOLUME: 1), PP 226-233. [ONLINE] DOI: 10.1109/IROS.1995.525801.
- .YANIV ALTSHULER, VLADIMIR YANOVSKI AND ISRAEL A. WAGNER (2008) "COOPERATIVE CLEANERS: A STUDY IN ANT ROBOTICS" THE INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH, VOL. 27 NO. 1127-151
- OWEN HOLLAND (2002),"THE FIRST BIOLOGICALLY INSPIRED ROBOT", ROBOTICA (VOLUME 21), ISSUE 04, PP 351-356. [ONLINE] DOI: 10.1017/S0263574703004971.
- FINCH HARDWARE (2014) SPECIFICATION, AVAILABLE AT <http://www.finchrobot.com/finch-hardware>
- BOSTON DYNAMICS ROBOTS CHEETAH AND SANDFLEA SPECIFICATION, AVAILABLE AT <http://www.bostondynamics.com/>
- SAMUEL GIBBS (2013)," WHAT IS BOSTON DYNAMICS AND WHY DOES GOOGLE WANT ROBOTS?", AVAILABLE AT [HTTP://WWW.THEGUARDIAN.COM/TECHNOLOGY/2013/DEC/17/GOOGLE-BOSTON-DYNAMICS-ROBOTS-ATLAS-BIGDOG-CHEETAH](http://www.theguardian.com/technology/2013/dec/17/google-boston-dynamics-robots-atlas-bigdog-cheetah)
- WALTER J FREEMAN (2003), "CJ. W.GREY WALTER: BIOGRAPHICAL ESSAY", ENCYCLOPEDIA OF COGNITIVE SCIENCE, PP 537-539.
- ANDREW ENGLISH (2014), "AUTONOMOUS CARS – IS THIS THE END OF DRIVING?" THE TELEGRAPH 16 JANUARY2014. AVAILABLE AT <http://www.telegraph.co.uk/motoring/road-safety/10570935/Autonomous-cars-is-this-the-end-of-driving.html>

Appendix A Personal Reflection

A.1 Reflection on Project

When we try to create or program robots we think of them doing stuff like we do. We try to share our logics and try to implement it in a finch. This also gave the idea of humanoids and since then many people and industries are working on humanoids. Creating something half as good as humans could require lot of research and cost lot of money. But there are also many simple species from where many algorithms originated and which are also comparatively more achievable in terms of both resources and cost. This simple idea created an excitement in my mind for working and researching in the area. Also teaching robots about team work is not only beneficial today but it will be beneficial in future too when there will be many robots who might need some common protocol for working in the real world.

I started researching on the topic through internet and library. I found lot of people already working on the field of co-operative mobile robotics but I also learned that the field is not very popular when compared to other fields due to lack of invested interest.

Finding a shortest route problem is the application I choose due to its complexity. I studied that finding the shortest route problem is a non-polynomial hard problem and still there isn't any proper solution to the problem except hit and trial. Travelling sales man problem also tries to focus on the same problem with no solution. Therefore in order to better understand the problem and understand why it is so hard to find the shortest route, I attempted this problem. I faced the same problem and I attempted ants and ACO (Ant Colony Optimization) inspired my own random route search algorithm for solving the problem. This algorithms also uses hit and trial but with the help of co-operative mobile robotics we can hit many solutions at a time which will provide us with better chances of success. More finches will run, more data we will have to analyze and lesser time it will take to find the shorter route. Since I was limited by one finch I made it to run many times in order to collect some data.

Some of the technical problem I took long time to solve was figuring out a right way to track the path moved by finch. It was very important for me to record path correctly otherwise none of my aim or objective could be achieved. Feedback from my tutor helped me look at my project more practically and also helped me find practical new solutions to the problems.

Another problem which I faced was that finch was not aware of its location. It needs to be experimented carefully. Every time starting position needs to be the same and wires could also be the influencing factor. I couldn't get time to use and explore raspberry pi this time but next time I would like to use raspberry pi for better results.

A.2 Personal Reflection

It was the first individual project I did and I learned lot of things from this project. Although my project is not exactly individual project. Me and my group partner Sangmok were one day discussing about the field of co-operative mobile robotics and that's when we decided to make this topic as our dissertation topic. Initially we planned to work together on some different algorithms but later through our tutor meetings we realized this would make most of mine and his work similar which might be hard to distinguish. Therefore we decided to work on one algorithm each in the common field and work on it individually. We then planned we will either compare our results or combine our algorithms depending on the types of algorithm.

I faced lot of problems in this project and I also learned to find solutions to those problems. One of the problem I faced includes pressure of finishing the project few weeks before the deadline. This could create problems as you are not left with enough time to test and make changes if anything goes wrong. It would be better if programming part is finished at least 2 months before the deadline for any final year project (FYP) sized project. It is also important to start working early as background research and reading journals takes lot of time. It is important to have complete idea of what we want to do before starting programming. Something like rough use case diagram helped me to stay on track. I also kept taking notes and collected all the relevant information in on place. This helped me stay pretty much organized.

Regular tutor meetings were very beneficial as getting regular feedback on the work we are doing could help us correct our self from beginning only.

I also learnt it is important to set up many mini goals for finishing the project which I'll definitely try doing on my next project. This not only keeps us organized but also helps us completing everything on time. After task 1 next task was task 2 which is the submission of the dissertation. This gives lot of time for distraction therefore mini goals could really help in staying on track.

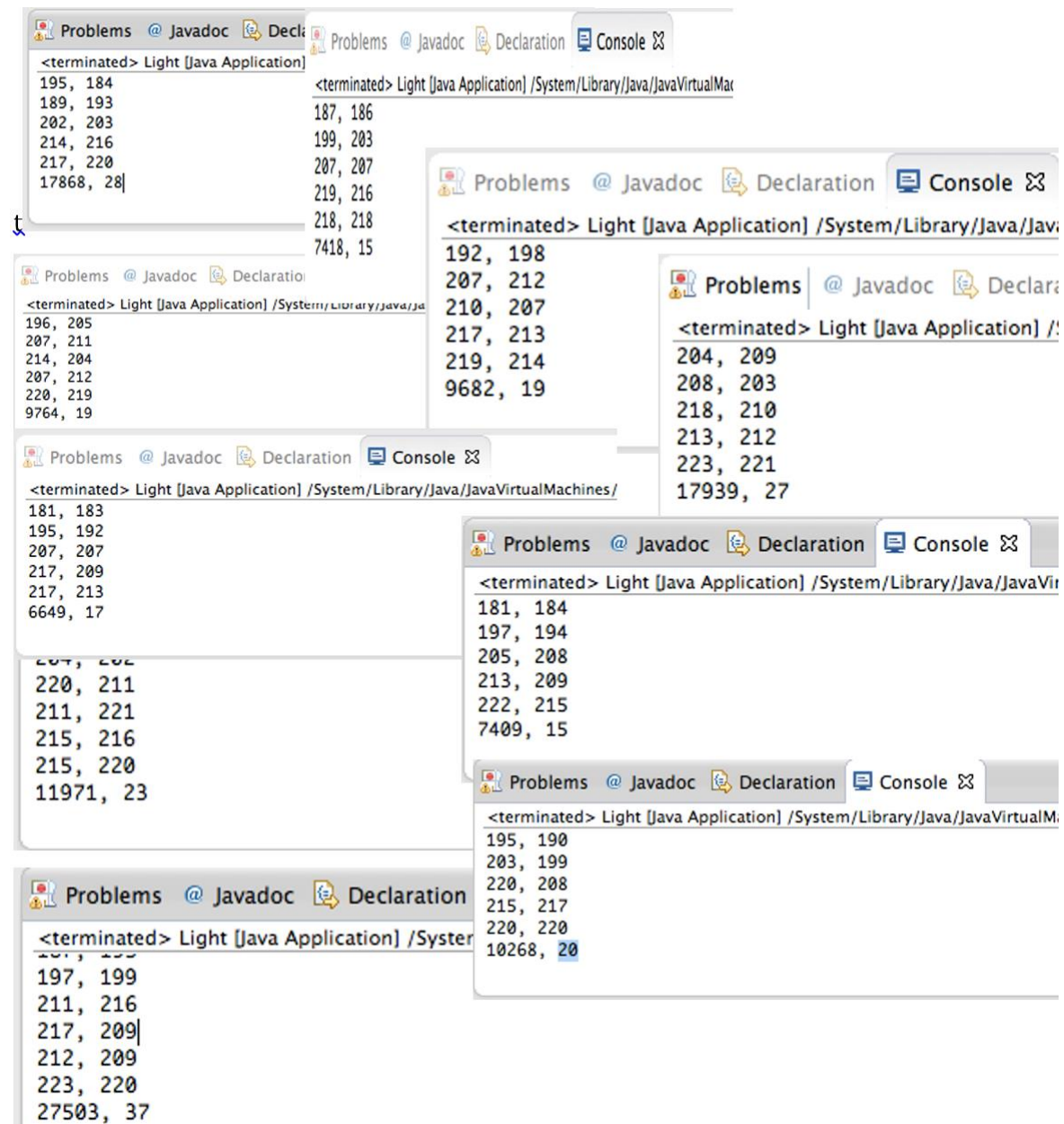
Appendix B Appendices

B.1 Data set

- 20 iterations raw data

Straight,Right,Left,Right,Straight,Straight,Back,Straight,Back./11.452,5.28,0.16,0.544,2.912,54.332,0.224,0.064,0.512+/
Straight,Back,Right,Back,Straight,Back,Straight,Straight,Right,Left,Back,Back,Right,Straight,Right,Left,Right,Left,Straight,
Back,Back,Right,Back,Straight,Left,Left,Right,Left,Back,Right,Left./
11.036,0.224,2.912,6.016,0.448,0.064,0.064,1.664,3.808,1.024,2.912,10.176,2.912,2.912,3.264,0.256,0.16,0.128,5.888,1.184,1.6
32,2.912,3.488,0.224,2.912,3.484,0.16,0.16,2.912,2.912,47.872+/
Straight,Back,Back,Straight,Back,Straight,Right,Left,Left,Left,Right,Left,Straight,Right,Left,Left,Right,Left,Right,Straight
,Right,Left,Right,Back,Straight,Right,Left,Right,Left,Left,Right,Left,Right,Straight,Straight,Back,Left./
11.324,0.224,0.256,0.224,0.128,0.032,5.44,0.992,0.16,0.832,0.032,1.408,11.548,10.272,0.224,0.16,0.544,0.032,0.128,2.912,6.17
2,0.896,0.032,13.12,0.224,21.184,0.128,0.256,0.128,1.888,0.032,0.224,0.032,2.912,16.672,0.256,11.744+/
Straight,Left,Left,Right,Left,Back./10.648,2.912,14.624,0.032,0.032,2.912+/
Straight,Back,Straight,Back,Back,Straight,Back,Straight,Left,Left./
11.132,0.224,0.032,0.032,1.408,0.224,0.032,0.128,2.912,16.0+/
Straight,Back,Left,Right,Left,Left,Right,Left,Back,Left,Right,Left,Straight,Back,Straight,Left,Left,Right,Straight,Straight.
/11.516,1.376,16.832,0.16,0.224,1.92,0.608,0.032,2.912,5.248,0.64,0.064,12.96,0.8,0.416,2.912,9.248,0.224,2.912,10.656+/
Straight,Back,Straight,Left,Left./10.844,0.224,0.32,2.912,21.568+/
Straight,Back,Straight,Back,Straight,Straight,Right,Left,Right,Left,Straight,Back,Straight,Straight,Back,Right,Left,Right,Le
ft,Back,Back,Straight,Back,Straight,Left,Right,Left,Right,Straight./
11.228,0.128,0.032,0.128,0.032,1.504,10.176,1.472,0.256,0.64,6.048,1.024,0.032,1.376,0.064,2.912,1.984,0.128,0.512,2.912,18.
848,0.16,1.216,0.704,2.912,0.448,0.224,0.032,2.912+/
10.46,0.224,0.032,2.912,5.76,2.912,7.968,4.416+/
Straight,Back,Left,Right,Left,Right,Straight,Right,Left,Right,Left,Right,Left,Right,Left,Back,Left,Right,Left,Right,Left,Str
aight,Back,Straight,Back,Back,Straight,Back,Right,Back,Straight,Right,Straight,Straight,Back,Left,Right,Left,Left,Back,Left,
Left,Right,Left,Left,Right,Back,Straight,Left,Straight,Back,Straight,Back,Left,Right,Left,Right,Left,Right,Left,Back,Right,S
traight,Right,Left,Left./
10.556,0.416,13.44,0.128,0.416,0.032,2.912,5.504,0.16,0.32,0.032,0.032,1.888,0.032,1.184,2.912,4.992,0.16,0.256,0.256,1.184,
12.864,0.028,0.16,0.416,1.532,0.224,0.512,2.944,6.016,0.256,3.456,2.912,2.368,0.256,4.544,0.576,0.032,0.544,2.912,5.184,0.06
4,0.256,0.64,1.312,0.64,13.792,0.8,2.912,5.344,0.256,0.064,0.032,13.28,0.224,0.064,0.16,1.152,0.064,0.992,2.912,2.912,2.912,
9.376,0.256,0.256+/
Straight,Back,Straight,Straight,Back,Back,Straight,Back,Straight,Right,Straight,Right,Left,Right,Left,Back,Left,Right,Straig
ht,Straight,Back,Back,Back,Left,Back,Back,Straight,Back,Right,Left,Left./
10.936,0.224,0.16,1.312,0.736,1.376,0.224,0.032,0.224,8.544,2.912,4.48,0.224,0.512,0.032,2.912,5.312,1.28,2.912,3.424,0.512,
0.064,0.544,3.968,2.912,1.728,0.224,0.512,2.912,16.96,0.384+/
Straight,Back,Straight,Left,Straight,Back,Straight,Left,Straight,Back,Right,Left,Right,Left,Right./
10.876,0.224,0.416,2.944,5.088,0.128,2.912,5.824,0.352,2.912,9.984,0.16,0.448,0.32+/
Straight,Back,Right,Back,Straight,Back,Right,Back,Straight,Back,Left,Right,Back,Left,Right,Straight,Straight,Back,Straight,L
eft,Left./
11.036,0.224,2.912,5.856,0.256,0.064,2.912,5.536,0.064,0.96,4.864,1.12,5.792,7.168,1.088,2.912,4.768,0.224,0.448,2.912,14.36
8+/
Straight,Back,Straight,Straight,Left,Back,Back./11.132,0.512,0.032,1.504,2.912,2.912,5.692+/
Straight,Back,Straight,Straight,Left,Left./11.036,0.512,0.032,1.248,2.912,13.376+/
Straight,Back,Straight,Back,Left,Right,Left,Right,Straight,Left,Left,Back,Back./
10.652,0.16,0.128,0.032,7.584,0.224,0.512,0.128,2.912,2.912,4.32,2.912,10.08+/
10.744,0.224,0.224,2.912,2.912,6.624+/
Straight,Back,Straight,Left,Back,Right,Straight,Straight,Right,Left,Right,Left,Back,Right,Left,Right,Back,Left,Right,Left,Ri
ght,Back,Straight,Back,Straight,Straight,Back,Right,Back,Right,Left,Right,Left,Straight,Back,Right,Left,Right,Left,Back,Righ
t,Straight,Left,Back,Back,Right,Left,Back,Back./
11.036,0.16,0.224,2.912,2.912,2.912,2.912,6.72,6.112,0.064,0.416,0.736,2.912,2.912,8.252,0.156,16.832,8.896,0.16,0.16,0.064,
7.968,0.16,0.896,0.032,2.016,0.928,2.912,6.688,2.912,6.816,0.128,0.352,5.248,0.032,2.912,1.504,0.992,0.128,2.912,2.912,2.912
,2.912,2.912,7.2,2.912,4.992,2.912,11.712+/
11.128,0.224,0.512,2.912,10.016,0.124,0.032,2.912,9.984+/
11.1,0.64,0.128,2.912,9.44+/
Straight,Back,Straight,Left,Left./

- Follow Light 10 iterations screenshot



B.2 CODE

- Shortest Distance index code, class updatedpheromone.

```
package ACO;

import java.util.ArrayList;

public class updatedpheromone {

    public int shortestdistance(ArrayList<Double> data){
        double min = Double.MAX_VALUE;
        int index = -1;
        for (int i = 0; i < data.size(); i++) {
            Double f = data.get(i);
            if (Double.compare(f.floatValue(), min) < 0) {
                min = f.floatValue();
                index = i;
            }
        }
        System.out.println("index is:"+index);
        return index;
    }
}
```

- Server Class

```
InetAddress computerAddr = null;
//Now store the local computer's name
try{
    computerAddr = InetAddress.getLocalHost();
}
catch(UnknownHostException e){
    System.out.println(e);
}

System.out.println("The address of this computer is.. " + computerAddr.getHostName());

// Now set up the server socket on port 4444 on the local machine
// Change the port so that someone else does not connect to it

ServerSocket serverSocket = null;
try {
    serverSocket = new ServerSocket(4444);
} catch (IOException e) {
    System.err.println("Could not listen on port: 4444.");
    System.exit(1);
}
System.out.println("Server up and waiting");
// Wait for client connection
// When a client connects, make the link and carry on
Socket clientSocket = null;
try {
    clientSocket = serverSocket.accept();
} catch (IOException e) {
    System.err.println("Accept failed.");
    System.exit(1);
}
// Connect the input and the output to and from the socket
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
```

B.3 Running Program Instruction

For making finch explore new paths and collect data.

- Run ACO main class and finch will start exploring.
- Once finch ran enough time to collect travel data go back to ACO main class and uncomment following codes
 - `ScanningRoutes k = new ScanningRoutes();`
 - `k.getdir();`
 - `k.sumofdistance();`
- Now comment out the following code and run the program
 - `path p = new path();`
 - `p.start();`
- Once program writes the shortest distance file in the computer run server class from server package.
- While server class running, run client file.
- Next go to second robo package and run robo2main class and finch will just follow the shortest route.